1,判断用户登陆状态

- 目的: 能够使用系统中的方法判断用户是否是登陆状态
- 方式一: 使用is_authenticated属性
- 操作流程:
  - 1, 根据前端访问的页面编写子路由

```
from django.conf.urls import url
from . import views

urlpatterns = [
    ...
    url(r'^info/$',views.UserCenterView.as_view()),
]
```

  - 2,编写类视图

```
class UserCenterView(View):
    def get(self,request):
        #1,判断用户是否登陆了
        if request.user.is_authenticated:
            return render(request, 'user_center_info.html')
        else:
            response = redirect("/login")
            response.delete_cookie("username")
            return response
```

- 方式二: login_required()装饰器
  - 1, 在子路由中,装饰路径

```
from django.conf.urls import url
from . import views
from django.contrib.auth.decorators import login_required

urlpatterns = [
    ...
     url(r'^info/$',login_required(views.UserCenterView.as_view())),
]
```

- 2,类视图编写

```
class UserCenterView(View):
    def get(self,request):
        ...
        #方式二:
        return render(request, 'user_center_info.html')
```

- 3,配置login_url

```
#用户若是没有登陆,到这来
LOGIN_URL = "/login/"
```

- 方式三: 扩展系统中的LoginRequiredMixin
  - 1,自定义类继承系统的类

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views import View

class MyLoginRequiredMixinView(LoginRequiredMixin,View):
    login_url = "/login/"
```

  - 2,类视图继承自我们自己编写的登陆类

```python
class UserCenterView(MyLoginRequiredMixinView):
    def get(self,request):
        return render(request,
'user_center_info.html')
```

2,qq登陆介绍

- 目的: 能够知道用户登陆的作用
- 作用: 首先qq拥有大量用户的,只要绑定过,下次就可以直接使用qq登陆了,方便用户体验

3,qq登陆文档

- 目的: 能够通过文档阅读, 获取到openid的流程
- 接口文档:
- 1, 获取Code
    - 请求路径:https://graph.qq.com/oauth2.0/authorize
    - 请求方式:GET
    - 请求参数:response_type, client_id, redirect_uri, state
    - 返回响应:code, state
        - 测 试 :https://graph.qq.com/oauth2.0/authorize?response_type=code&client_id=101518219&redirect_uri=http://www.meiduo.site:8000/oauth_callback&state=/
- 2,获取Access Token
    - 请求路径:https://graph.qq.com/oauth2.0/token
    - 请求方式:GET
    - 请求参数: grant_type, client_id, client_secret,code,redirect_uri
    - 返回响应:access_token
- 3,获取openid
    - 请求路径:https://graph.qq.com/oauth2.0/me
    - 请求方式:GET
    - 请求参数: access_token

- 返回响应:openid
    - openid是qq服务器上唯一对应用户身份的标识,

4,qq登陆SDK测试

- 目的: 能够使用qq登陆的sdk实现openid的获取
- 获取流程:
    - 1,安装qq工具类
        - pip install QQLoginTool
    - 2,将qq的相应配置,设置到dev.py中

```
QQ_CLIENT_ID = '101518219'
QQ_CLIENT_SECRET = 
'418d84ebdc7241efb79536886ae95224'
QQ_REDIRECT_URI = 
'http://www.meiduo.site:8000/oauth_callback'
```

    - 3,进入到终端中,输入python manage.py shell
    - 4,导入包
        - from QQLoginTool.QQtool import OAuthQQ
        - from django.conf import settings
    - 5,初始化OAuthQQ对象
        - oauth = OAuthQQ(client_id=settings.QQ_CLIENT_ID, client_secret=settings.QQ_CLIENT_SECRET, redirect_uri=settings.QQ_REDIRECT_URI, state=next)
    - 6,获取qq登陆页面
        - login_url = oauth.get_qq_url()
        - print(login_url)
    - 7,将login_url拷贝到浏览器的地址,使用qq扫码登陆,获取code
        - code: 148A0F70A6DDC209CFE644D5F3FBB094
    - 8,通过Authorization Code获取Access Token
        - access_token = oauth.get_access_token(code)
    - 9,通过Access Token获取OpenID
        - openid = oauth.get_open_id(access_token)

        - print(openid)

5,qq登陆模型类

- 目的: 能够创建qq用户模型类,并理解user和openid两个字段即可
- 操作流程:
    - 1, 创建了oatuh的子应用
    - 2, 编写了模型类,继承自了BaseModel

```python
from django.db import models
from meiduo_mall.utils.models import BaseModel

class OAuthQQUser(BaseModel):
    user = models.ForeignKey("users.User",
                             on_delete=models.CASCADE,
                             verbose_name="关联的美多用户")
    openid = models.CharField(max_length=64,verbose_name="openid")

    class Meta:
        db_table = "tb_oauth_qq"

```

    - 3,定义工具的模型类BaseModel

```
from django.db import models

class BaseModel(models.Model):
    create_time = models.DateTimeField(auto_now_add=True,verbose_name="创建时间")
    update_time = models.DateTimeField(auto_now=True,verbose_name="修改时间")

    class Meta:
        abstract = True #表示用户被继承,不会迁移生成数据库所对应的具体的表
```

- 4,迁移

6,qq登陆界面返回

- 目的: 能够编写类视图获取qq登陆页面
- 操作流程:
  - 1, 根据前端页面编写子应用

```
from django.conf.urls import url
from . import views

urlpatterns = [
  url(r'^qq/login/$',views.OAuthQQLoginView.as_view())
]
```

  - 2,编写根路由

```
urlpatterns = [
    ...
    url(r'^', include('oauth.urls',namespace="oauth")),
]

```

- 3,类视图

```python
class OAuthQQLoginView(View):
    def get(self,request):
        #1,获取参数
        state = request.GET.get("next","/")

        #2,创建OAuthQQ对象
        oauth_qq = OAuthQQ(client_id=settings.QQ_CLIENT_ID,
                client_secret=settings.QQ_CLIENT_SECRET,
                redirect_uri=settings.QQ_REDIRECT_URI,
                state=state)

        #3,获取qq登陆页面
        login_url = oauth_qq.get_qq_url()

        #4,返回
        return http.JsonResponse({"login_url":login_url})
```

7,openid获取

- 目的: 能够通过扫码之后的code,获取到最终的openid
- 操作流程:
    - 1, 将oauth_callback.html移动到templates
    - 2,根据扫码之后的接口, 编写子路由

```
from django.conf.urls import url
from . import views

urlpatterns = [
    ...

 url(r'^oauth_callback/$',views.OAuthUserView.as_view()),
]
```

- 3,编写类视图

```
class OAuthUserView(View):
    def get(self,request):
        #1,获取参数,code
        code = request.GET.get("code")
        state = request.GET.get("state","/")

        if not code:
            return http.HttpResponseForbidden("code丢了")

        #2,通过code换取access_token
        oauth_qq = OAuthQQ(client_id=settings.QQ_CLIENT_ID,
                           client_secret=settings.QQ_CLIENT_SECRET,
                           redirect_uri=settings.QQ_REDIRECT_URI,
                           state=state)
        access_token = oauth_qq.get_access_token(code)

        #3,通过access_token换取openid
        openid = oauth_qq.get_open_id(access_token)

        #4,返回响应
        return http.HttpResponse("openid=%s"%openid)
```

8,itsdangerous

- 目的:能够参考文档,加密数据,并且设计数据的有效期
- JSONWebSignatureSerializer:
    - 特点: 不支持有效期的设置
- TimedJSONWebSignatureSerializer
    - 特点:支持有效期的设置
    - 使用

```
In [8]: from itsdangerous import TimedJSONWebSignatureSerializer as TJWSSerializer

In [9]: s2 = TJWSSerializer(secret_key='xixi',expires_in=30)

In [10]: result1 = s2.dumps({"name":"laowang"})

In [11]: result2 = s2.loads(result1)

In [12]: result2

Out[12]: {'name': 'laowang'}

```

```
In [13]: result2 = s2.loads(result1)



---------------------------------------------------------------------------
SignatureExpired
 Traceback (most recent call last)
<ipython-input-13-c501b54192fb> in <module>
----> 1 result2 = s2.loads(result1)

~/.virtualenvs/django_py3/lib/python3.6/site-packages/itsdangerous/jws.py in loads(self, s, salt, return_header)
    203                         "Signature expired",
    204                         payload=payload,
--> 205                         date_signed=self.get_issue_date(header),
    206                     )
    207

SignatureExpired: Signature expired

```

9,非初次授权绑定用户

- 目的: 如果非初次授权能够设置用户的状态保持信息
- 操作流程:

```
class OAuthUserView(View):
    def get(self,request):
        .....

        #3,通过access_token换取openid
        openid = oauth_qq.get_open_id(access_token)

        #4,根据openid,取到qq用户的对象
        try:
            oauth_qq_user = OAuthQQUser.objects.get(openid=openid)

        except OAuthQQUser.DoesNotExist:
            #初次授权
            return render(request,'oauth_callback.html')
        else:
            #5,非初次授权
            user = oauth_qq_user.user

            #5,1状态保持
            login(request,user)
            request.session.set_expiry(3600*24*2)

            #5,2,返回响应
            response = redirect("/")
            response.set_cookie("username",user.username)
            return response
```

11, 授权页面获取,携带加密openid

- 目的: 能够渲染页面的时候,添加openid
- 操作流程:
    - 1, 定义加密,解密方法(oauth/utils.py)

```python
from itsdangerous import TimedJSONWebSignatureSerializer as TJSWSSerializer

#1,加密openid
def generate_sign_openid(openid):

    #1,创建TJSWSSerializer对象
    serializer = TJSWSSerializer(secret_key="oauth",expires_in=300)

    #2,加密openid
    sign_openid = serializer.dumps({"openid":openid})

    #3,返回结果
    return sign_openid.decode()

#2,解密openid
def decode_sign_openid(data):
    # 1,创建TJSWSSerializer对象
    serializer = TJSWSSerializer(secret_key="oauth", expires_in=300)

    # 2,加密openid
    try:
        data_dict = serializer.loads(data)
    except Exception as e:
        return None

    # 3,返回结果
    return data_dict.get("openid")
```

- 2,渲染页面

```python
class OAuthUserView(View):
    def get(self,request):
        ...
        #4,根据openid,取到qq用户的对象
        try:
            oauth_qq_user = OAuthQQUser.objects.get(openid=openid)

        except OAuthQQUser.DoesNotExist:
            #初次授权
            sign_openid = generate_sign_openid(openid) 修改的位置
            return render(request,'oauth_callback.html',
                                      context={"token":sign_openid})
        else:
            ...
```

12, 授权参数校验

- 目的: 能够对传递进来的授权参数做校验

```
class OAuthUserView(View):
    ...
    def post(self,request):
        #1,获取参数
        sign_openid = request.POST.get("access_token")
        mobile = request.POST.get("mobile")
        pwd = request.POST.get("pwd")
        sms_code = request.POST.get("sms_code")

        #2,校验参数
        #2,1为空校验
        if not all([sign_openid,mobile,pwd,sms_code]):
            return http.HttpResponseForbidden("参数不全")

        #2,2 校验sign_openid是否证券
        openid = decode_sign_openid(sign_openid)
        if not openid:
            return http.HttpResponseForbidden("openid过期")

        #2,3 校验手机号的格式
        if not re.match(r'^1[3-9]\d{9}$',mobile):
            return http.HttpResponseForbidden("手机号格式有误")

        #2,4 校验密码的格式
        if not re.match(r'^[0-9a-zA-Z]{8,20}$', pwd):
            return http.HttpResponseForbidden("密码格式有误")

        #2,5 校验短信验证码的正确性
        redis_conn = get_redis_connection("code")
        redis_sms_code = redis_conn.get("sms_code_%s"%mobile)

        if not redis_sms_code:
            return http.HttpResponseForbidden("短信验证码已过期")

        if sms_code != redis_sms_code.decode():
            return http.HttpResponseForbidden("短信验证码填写错误")

        #3,判断是否存在美多用户,如果存在直接绑定,如果不存在直接创建用户再绑定

        #4,状态保持

        #5,返回到首页中
        pass

```

13, 授权用户绑定

- 目的: 能够知道, 存在或者不存在美多商城用户的授权绑定
- 操作流程:

```
class OAuthUserView(View):
    ...

    def post(self,request):
       ...
        #3,判断是否存在美多用户,如果存在直接绑定,如果不存在直接创建用户再绑定
        try:
            user = User.objects.get(mobile=mobile)
        except User.DoesNotExist:
            #3,1创建美多用户
            user = User.objects.create_user(username=mobile,password=pwd,mobile=mobile)

            #3,2绑定美多用户和qq用户
            OAuthQQUser.objects.create(openid=openid,user=user)

            # 3.3,状态保持
            login(request, user)
            request.session.set_expiry(3600 * 24 * 2)

            # 3.4,返回到首页中
            response = redirect("/")
            response.set_cookie("username", user.username, max_age=3600 * 24 * 2)
            return response
        else:
            #4.1校验密码正确性
            if not user.check_password(pwd):
                return http.HttpResponseForbidden("密码错误")

            #4,2绑定美多用户和qq用户
            OAuthQQUser.objects.create(openid=openid,user=user)

            #4.3,状态保持
            login(request,user)
            request.session.set_expiry(3600*24*2)

            #4.4,返回到首页中
            response = redirect("/")
            response.set_cookie("username",user.username,max_age=3600*24*2)
            return response

```